

Team T

1. Hemanth Thathireddy (A20525346)
2. Satya Dineswara Reddy Satti (A20528639)
3. Sai Manasa Basani (A20544372)
4. Alavalapati Veera manohar Reddy (A20526070)
5. Ying Zhang (A20547843)

Part 1 : Advanced window functions

1. **SQL Statement:** Write a query to compute for the **FIRST_VALUE()** given the above dataset and return the value along with the entire row.

SQL Query: `SELECT *,
FIRST_VALUE(row_num) OVER (ORDER BY Salary ASC) AS FirstValue
FROM
SalaryEmployee1 limit 1;`

Result:

```
-- 1  
SELECT *,  
FIRST_VALUE(row_num) OVER (ORDER BY Salary ASC) AS FirstValue  
FROM  
SalaryEmployee1 limit 1;
```

row_num	first_name	last_name	salary	FirstValue
1	Karen	Colmenares	2500	1

The screenshot shows the SSMS interface with the query results. The results grid displays a single row with the following data:
row_num: 1
first_name: Karen
last_name: Colmenares
salary: 2500
FirstValue: 1

2. **SQL Statement:** Write a query to compute for the **LAST_VALUE()** and return the value along with the entire row.

SQL Query:

```
SELECT *,  
LAST_VALUE (row_num) OVER(ORDER BY salary DESC) as last_value_answer  
FROM SalaryEmployee1 limit 1;
```

Result:

The screenshot shows the SSMS interface with the following details:

- Schemas:** The current schema is set to **SalaryDB**.
- Query Editor:** The query window contains the following T-SQL code:


```
7   -- 2
8
9 • SELECT *,  
10    LAST_VALUE (row_num) OVER(ORDER BY salary DESC) as last_value_answer  
11  FROM SalaryEmployee1 limit 1;  
12
```
- Result Grid:** The results are displayed in a grid with the following columns:

row_num	first_name	last_name	salary	last_value_answer
15	Charles	Johnson	6200	15
- Right Panel:** Various tools and options are available, including **Result Grid**, **Form Editor**, **Field Types**, **Query Stats**, and **Execution Plan**.

3. **SQL Statement:** Write a query to compute for **LEAD(2)** for Guy and return the value along with the Guy's row.

SQL Query:

```
select * from (select *, lead(row_num,2)  
over(order by salary) lead2_guy from SalaryEmployee1) as table1  
where first_name='Guy';
```

Result:

```

13 -- 3
14
15 • select * from (select *, lead(row_num,2)
16 over(order by salary) lead2_guy from SalaryEmployee1) as table1
17 where first_name='Guy';
18

```

row_num	first_name	last_name	salary	lead2_guy
2	Guy	Himuro	2600	4

Action Output

4. **SQL Statement:** Write a query to compute for LAG(4) for Pat and return value along with Pat's row.

SQL Query: `SELECT * FROM(`

```

        SELECT row_num, first_name, last_name, salary,
LAG (row_num, 4) OVER (ORDER BY Salary) lag4_Pat FROM SalaryEmployee1) as
table1
WHERE first_name = "Pat";

```

Result:

The screenshot shows the SSMS interface with the following details:

- Schemas:** The current schema is set to **SalaryDB**.
- Query Editor:** The query window contains the following T-SQL code:


```

20
21 • SELECT * FROM(
22   SELECT row_num, first_name, last_name, salary,
23   LAG (row_num, 4) OVER (ORDER BY Salary) lag4_Pat FROM SalaryEmployee1) as table1
24 WHERE first_name = "Pat";
25
      
```
- Result Grid:** The results show one row for the employee Pat Fay.

row_num	first_name	last_name	salary	lag4_Pat
14	Pat	Fay	6000	10

- Right Panel:** Various tools and options are available, including Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.

5. **SQL Statement:** Write a query to compute the **RANK()** and **DENSE_RANK()** and return the entire dataset, including the rank and dense rank for each employee.

SQL Query:

```

SELECT *,
RANK() OVER (ORDER BY Salary) as ranks,
DENSE_RANK() OVER (ORDER BY Salary) as dense_ranks
FROM SalaryEmployee1;
      
```

Result:

```

27
28 •  SELECT *, 
29      RANK() OVER (ORDER BY Salary) as ranks,
30      DENSE_RANK() OVER (ORDER BY Salary) as dense_ranks
31  FROM SalaryEmployee1;
32

```

row_num	first_name	last_name	salary	ranks	dense_ranks
1	Karen	Colmenares	2500	1	1
2	Guy	Himuro	2600	2	2
3	Irene	Millineni	2700	3	3
4	Sigal	Tobias	2800	4	4
5	Shelli	Baida	2900	5	5
6	Alexander	Khoo	3100	6	6
7	Britney	Everett	3900	7	7
8	Sarah	Bell	4000	8	8
9	Diana	Lorentz	4200	9	9
10	Jennifer	Whalen	4400	10	10
11	David	Austin	4800	11	11
12	Valli	Pataballa	4800	11	11
13	Bruce	Ernst	6000	13	12
14	Pat	Fay	6000	13	12
15	Charles	Johnson	6200	15	13

Result 62 Read Only

6. **SQL Statement:** Write a query to compute the **RANK()** and **DENSE_RANK()** but only return Valli's and Bruce's rank and dense rank.

SQL Query: `SELECT *,
RANK() OVER (ORDER BY Salary) as ranks,
DENSE_RANK() OVER (ORDER BY Salary) as dense_ranks
FROM SalaryEmployee1 as table1
where first_name IN ('Valli', 'Bruce');`

Result:

```

34
35 •   SELECT *, 
36       RANK() OVER (ORDER BY Salary) as ranks,
37       DENSE_RANK() OVER (ORDER BY Salary) as dense_ranks
38   FROM SalaryEmployee1 as table1
39   where first_name IN ('Valli', 'Bruce');
40
41 -- 7

```

The screenshot shows the SSMS interface with the following details:

- Schemas:** The current schema is set to **SalaryDB**.
- Result Grid:** Displays the results of the executed query. The columns are **row_num**, **first_name**, **last_name**, **salary**, **ranks**, and **dense_ranks**. The data shows two rows for Valli and Bruce.
- Execution Plan:** A vertical sidebar on the right contains icons for Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.
- Status Bar:** Shows "Result 63" and "Read Only".

row_num	first_name	last_name	salary	ranks	dense_ranks
12	Valli	Pataballa	4800	1	1
13	Bruce	Ernst	6000	2	2

7. **SQL Statement:** Write a query to compute the **ROW_NUMBER()** for Irene and Sarah and only return the rows corresponding to them.

SQL Query: `SELECT * FROM(`
`SELECT row_num, first_name, last_name, salary,`
`ROW_NUMBER() OVER (ORDER BY Salary) as row_numbers`
`FROM SalaryEmployee1) as table1`
`WHERE first_name IN ('Irene', 'Sarah');`

Result:

The screenshot shows the SSMS interface with the following details:

- Schemas:** The current schema is set to **SalaryDB**.
- Query Editor:** The query window contains the following T-SQL code:


```

41 -- 7
42
43 • SELECT * FROM(
44   SELECT row_num, first_name, last_name, salary,
45   ROW_NUMBER() OVER (ORDER BY Salary) as row_numbers
46   FROM SalaryEmployee1) as table1
47 WHERE first_name IN ('Irene', 'Sarah');
48
      
```
- Result Grid:** The results show the following data:

	row_num	first_name	last_name	salary	row_numbers
3	Irene	Millilineni	2700	3	
8	Sarah	Bell	4000	8	
- Right Panel:** The sidebar on the right lists various tools and options:
 - Result Grid
 - Form Editor
 - Field Types
 - Query Stats
 - Execution Plan

8. **SQL Statement:** Write a query to compute the **PERCENT_RANK()** and return the entire dataset, including the percent rank for each employee. Format your **PERCENT_RANK()** values to 100%.

SQL Query:

```

SELECT row_num, first_name, last_name, salary,
RANK() OVER (ORDER BY Salary) as'Rank',
ROUND(ROUND(PERCENT_RANK()) OVER (ORDER BY Salary),2)*100,2
as'PercentRank (%)'
FROM SalaryEmployee1;
      
```

Result:

```

48
49      -- 8
50
51 •  SELECT row_num, first_name, last_name, salary,
52     RANK() OVER (ORDER BY Salary) as 'Rank',
53     ROUND (ROUND (PERCENT_RANK() OVER (ORDER BY Salary),2) *100,2) as 'PercentRank (%)'
54   FROM SalaryEmployee1;
55

```

	row_num	first_name	last_name	salary	Rank	PercentRank (%)
1	Karen	Colmenares	2500	1	1	0
2	Guy	Himuro	2600	2	7	7
3	Irene	Millineni	2700	3	14	14
4	Sigal	Tobias	2800	4	21	21
5	Shelli	Baida	2900	5	29	29
6	Alexander	Khoo	3100	6	36	36
7	Britney	Everett	3900	7	43	43
8	Sarah	Bell	4000	8	50	50
9	Diana	Lorentz	4200	9	57	57
10	Jennifer	Whalen	4400	10	64	64
11	David	Austin	4800	11	71	71
12	Valli	Pataballa	4800	11	71	71
13	Bruce	Ernst	6000	13	86	86
14	Pat	Fay	6000	13	86	86
15	Charles	Johnson	6200	15	100	100

Result 65 Read Only

9. **SQL Statement:** Write a query to compute the **CUME_DIST()** and return the entire dataset, including the percentage rank for each employee. Format your CUME_DIST() values to 2 decimal places.

SQL Query: `SELECT row_num, first_name, last_name, salary,
RANK() OVER (ORDER BY Salary) as 'Rank',
ROUND (CUME_DIST() OVER (ORDER BY Salary),2) as 'Cummulative Distribution'
FROM SalaryEmployee1;`

Result:

```

57
58 •  SELECT row_num, first_name, last_name, salary,
59      RANK() OVER (ORDER BY Salary) as 'Rank',
60      ROUND (CUME_DIST() OVER (ORDER BY Salary),2) as 'Cummulative Distribution'
61  FROM SalaryEmployee1;
62
63  -- 10
64

```

	row_num	first_name	last_name	salary	Rank	Cummulative Distribution...
1	1	Karen	Colmenares	2500	1	0.07
2	2	Guy	Himuro	2600	2	0.13
3	3	Irene	Millineni	2700	3	0.2
4	4	Sigal	Tobias	2800	4	0.27
5	5	Shelli	Baida	2900	5	0.33
6	6	Alexander	Khoo	3100	6	0.4
7	7	Britney	Everett	3900	7	0.47
8	8	Sarah	Bell	4000	8	0.53
9	9	Diana	Lorentz	4200	9	0.6
10	10	Jennifer	Whalen	4400	10	0.67
11	11	David	Austin	4800	11	0.8
12	12	Valli	Pataballa	4800	11	0.8
13	13	Bruce	Ernst	6000	13	0.93
14	14	Pat	Fay	6000	13	0.93
15	15	Charles	Johnson	6200	15	1

Result 66 Read Only

10. **SQL Statement:** Write a query to compute the **NTILE(4)** and return the entire dataset showing approximately equal groups/buckets.

SQL Query: `SELECT row_num, first_name, last_name, salary,
NTILE(4) OVER (ORDER BY Salary) as 'Cluster'
FROM SalaryEmployee1;`

Result:

```

61   FROM SalaryEmployee1;
62
63   -- 10
64
65 •  SELECT row_num, first_name, last_name, salary,
66   NTILE(4) OVER (ORDER BY Salary) as 'Cluster'
67   FROM SalaryEmployee1;
68

```

row_num	first_name	last_name	salary	Cluster
1	Karen	Colmenares	2500	1
2	Guy	Hiru	2600	1
3	Irene	Millineni	2700	1
4	Sigal	Tobias	2800	1
5	Shelli	Baida	2900	2
6	Alexander	Khoo	3100	2
7	Britney	Everett	3900	2
8	Sarah	Bell	4000	2
9	Diana	Lorentz	4200	3
10	Jennifer	Whalen	4400	3
11	David	Austin	4800	3
12	Valli	Pataballa	4800	3
13	Bruce	Ernst	6000	4
14	Pat	Fay	6000	4
15	Charles	Johnson	6200	4

Result 67 Read Only

Windowing Activity:

- SQL Statement:** Find a running total of covid admissions over three days (i.e., previous day, current day and next day)

SQL Query:

SELECT

```

yyyyymmdd,
SUM(admissions) OVER(ORDER BY yyyymmdd ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM coviddata;

```

Result:

```

SELECT
    yyyymmdd,
    SUM(admissions) OVER(ORDER BY yyyymmdd ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM coviddata;

```

The screenshot shows the SSMS interface with the following details:

- Schemas:** Shows the current schema is **SalaryDB**.
- Tables:** Shows tables like **coviddata**, **ReginalSales**, **SalaryEmployee1**.
- Result Grid:** Displays the results of the query, which is a 7-day moving average of admissions.
- Result Grid Headers:** **yyyymmdd** and **RunningTotal**.
- Results:** The grid contains 26 rows of data from May 24 to June 12, 2020.
- Execution Plan:** A small icon in the bottom right corner indicates the plan is read-only.

yyyymmdd	RunningTotal
2020-05-24	3046
2020-05-25	6262
2020-05-26	9701
2020-05-27	13503
2020-05-28	17648
2020-05-29	21738
2020-05-30	26099
2020-05-31	30645
2020-06-01	35363
2020-06-02	40290
2020-06-03	45368
2020-06-04	50745
2020-06-06	56690
2020-06-07	62870
2020-06-08	69223
2020-06-09	75905
2020-06-10	82897
2020-06-11	90188
2020-06-12	97744

II. SQL Statement: Find a 7-day moving average of covid patients in ICU.

SQL Query:

SELECT

```

date,yyyymmdd,state,ICU,
AVG(ICU) OVER (
    ORDER BY yyyymmdd,date
    ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS
seven_day_moving_avg_ICU
FROM
    coviddata;

```

Result:

The screenshot shows the SSMS interface with the following details:

- Schemas:** The current schema is set to **SalaryDB**.
- Query Editor:** The code area contains the following T-SQL query:


```

76 -- 12
77
78 • SELECT
79     date,yyyymmdd,state,ICU,
80     AVG(ICU) OVER (
81         ORDER BY yyyymmdd,date
82         ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS seven_day_moving_avg_ICU
83
84 FROM
85     coviddata;
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
      
```
- Result Grid:** The results of the query are displayed in a grid format. The columns are **date**, **yyyymmdd**, **state**, **ICU**, and **seven_day_moving_avg_ICU**. The data starts on May 24, 2020, and ends on June 10, 2020. The result count is 69.
- Result Grid Options:** On the right side of the result grid, there are several icons for different operations: **Result Grid**, **Form Editor**, **Field Types**, and **Query Stats**.

III. SQL Statement: Find a running total of covid deaths in each state

SQL Query:

SELECT

```

        date,yyyymmdd,state,ICU,
        AVG(ICU) OVER (
            ORDER BY yyyymmdd,date
            ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS
    seven_day_moving_avg_ICU
    FROM
        coviddata;
      
```

Result:

```

86 -- 13
87
88 • SELECT
89     date,yyyymmdd,state,hospital_deaths,
90     SUM(hospital_deaths) OVER (
91         PARTITION BY state
92         ORDER BY yyyymmdd) AS running_total_deaths
93
94 FROM
95     coviddata;
96

```

	date	yyyymmdd	state	hospital_deaths	running_total_deaths
1	24/05/20	2020-05-24	AL	389	389
2	01/06/20	2020-06-01	AL	608	997
3	02/06/20	2020-06-02	AL	643	1640
4	13/06/20	2020-06-13	AL	1088	2728
5	17/06/20	2020-06-17	AL	1265	3993
6	25/05/20	2020-05-25	AZ	409	409
7	03/06/20	2020-06-03	AZ	663	1072
8	11/06/20	2020-06-11	AZ	1016	2088
9	14/06/20	2020-06-14	AZ	1128	3216
10	18/06/20	2020-06-18	AZ	1333	4549
11	23/06/20	2020-06-23	AZ	1592	6141
12	25/06/20	2020-06-25	AZ	1739	7880
13	27/05/20	2020-05-27	DC	472	472
14	29/05/20	2020-05-29	DC	519	991
15	31/05/20	2020-05-31	DC	587	1578
16	06/06/20	2020-06-06	DC	775	2353
17	08/06/20	2020-06-08	DC	854	3207

Result 70 Read Only

Part 2 : OLAP Queries

- I. **SQL Statement:** What are the total sales of each product in each region and country?

SQL Query:

```

SELECT
    Product,
    Region,
    Country,
    SUM(Sales) AS total_sales
FROM
    ReginalSales
GROUP BY
    Product,
    Region,
    Country;

```

Result:

The screenshot shows the SSMS interface with the following details:

- Schemas:** The current schema is set to "SalaryDB".
- Query Editor:** A T-SQL query is displayed:


```

98 •  SELECT
99      Product,
100     Region,
101    Country,
102   SUM(Sales) AS total_sales
103  FROM
104    ReginalSales
105 GROUP BY
106    Product,
107    Region,
108    Country;
      
```
- Result Grid:** The results of the query are shown in a tabular format:

	Product	Region	Country	total_sales
A	Asia	China	4200	
A	Asia	Japan	800	
A	North America	Canada	3200	
A	North America	USA	8200	
A	Europe	Germany	300	
B	Europe	Germany	700	
B	Europe	France	8700	
B	Europe	Italy	400	
B	Australia	Australia	6000	
B	Australia	New Zealand	5500	
- Message Bar:** Shows "Result 71" and a "Read Only" status.
- Right Panel:** Contains icons for "Result Grid", "Form Editor", and "Field Types".

- II. **SQL Statement:** What are the total sales of each product in each region and country, including subtotals and the grand total?

SQL Query:

```
select Product,Region,Country,sum(Sales) totalsales from Regional sales group by Product,Region,country with rollup;
```

Result:

The screenshot shows a database management interface with the following details:

- Schemas:** Sales
- Tables:** regional
- Columns:** Product, Region, Country, City, Sales
- SQL Query:**

```
1 USE Sales;
2 • SELECT Product, Region, Country, SUM(Sales) as Total_Sales
3   from regional
4   Group by Product, Region, Country with ROLLUP;
```
- Result Grid:** A table showing the query results. The columns are Product, Region, Country, and Total_Sales. The data includes rows for products A and B across various regions and countries, with some rows being subtotal or grand total entries.
- Toolbar:** Includes icons for file operations, search, and export.
- Right Panel:** Icons for Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan. A "Read Only" status is also indicated.

Product	Region	Country	Total_Sales
A	Asia	China	4200
A	Asia	Japan	800
A	Asia	HULL	5000
A	Europe	Germany	300
A	Europe	HULL	300
A	North America	Canada	3200
A	North America	USA	8200
A	North America	HULL	11400
A	HULL	HULL	16700
B	Australia	Australia	6000
B	Australia	New Ze...	5500
B	Australia	HULL	11500
B	Europe	France	8700
B	Europe	Germany	700
B	Europe	Italy	400
B	Europe	HULL	9800
B	HULL	HULL	21300
	HULL	HULL	38000

- III. **SQL Statement:** What are the total sales generated by each product in each region and country, including all subtotals and grand totals?

SQL Query:

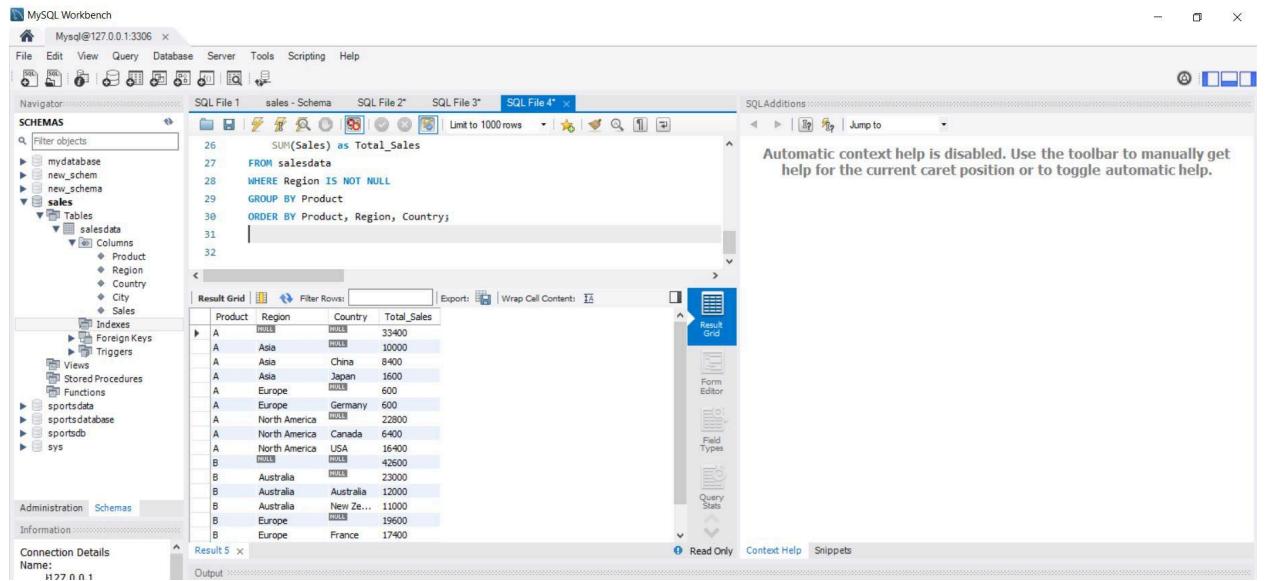
```
SELECT
    Product,
    Region,
    Country,
    SUM(Sales) as Total_Sales
FROM salesdata
GROUP BY Product, Region, Country
UNION ALL
-- Subquery for Product, Region level totals
SELECT
    Product,
    Region,
    NULL AS Country,
    SUM(Sales) as Total_Sales
FROM salesdata
WHERE Region IS NOT NULL
GROUP BY Product, Region
UNION ALL
-- Subquery for Product level totals
SELECT
    Product,
```

```

NULL AS Region,
NULL AS Country,
SUM(Sales) as Total_Sales
FROM salesdata
WHERE Region IS NOT NULL
GROUP BY Product
ORDER BY Product, Region, Country;

```

Result:



The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Includes icons for New Query, Open, Save, Print, Copy, Paste, Find, Replace, and others.
- Navigator:** Shows the database schema with the following structure:
 - SCHEMAS:** mydatabase, new_schem, new_schema, sales.
 - Tables:** salesdata (Columns: Product, Region, Country, Sales).
 - Indexes:**
 - Foreign Keys:**
 - Triggers:**
 - Views:**
 - Stored Procedures:**
 - Functions:**
 - sportsdata**
 - sportsdatabase**
 - sportddb**
 - sys**
- SQL Editor:** Contains the executed SQL query.
- Result Grid:** Displays the query results in a tabular format.
- Output:** Shows the result set with the following data:

Product	Region	Country	Total_Sales
A	NULL	NULL	33400
A	Asia	NULL	10000
A	Asia	China	8400
A	Asia	Japan	1600
A	Europe	NULL	600
A	Europe	Germany	600
A	North America	NULL	22800
A	North America	Canada	6400
A	North America	USA	16400
B	NULL	NULL	42600
B	Australia	NULL	23000
B	Australia	Australia	12000
B	Australia	New Ze...	11000
B	Europe	NULL	19600
B	Europe	France	17400

Team Contributions:

S.No	Name	Contribution(%)
1.	Hemanth Thathireddy	20%
2.	Satya Dineswara Reddy Setti	20%
3.	Sai Manasa Basani	20%
4.	Alavalapati Veera manohar Reddy	20%
5.	Ying Zhang	20%